มุมนักเล่น

Clipper

จเร เลิศสุดวิชัย

โค้ดบล็อก (Code Block) ตอนที่ 1

มารู้จักกับแบบของข้อมูลอีกชนิดที่ช่วยอำนวยความสะดวกและลดเวลาในการทำงานที่เหนือชั้นกว่ามาโคร

ผู้เขียนเคยกล่าวไว้แล้วว่าคลิปเปอร์ 5.0 นั้นเป็นภาษา xBASE ที่ค่อนข้างมีความอ่อนตัวกว่าดีเบสที่เป็นต้นแบบของคลิปเปอร์เอง นอกจากนั้นในคลิปเปอร์ 5.0 ยังมีสิ่งใหม่ ๆ ที่ภาษาคอมพิวเตอร์สมัยใหม่พึงมี เช่น โค้ดบล็อก (Code Block) ซึ่งถ้าเปรียบเทียบก็จะคล้ายคลึงกับพอยน์เตอร์ของฟังก์ชันในภาษาซี แต่มีคุณสมบัติพิเศษมากกว่า โดยในมุมคลิปเปอร์ตอนนี้ ผู้เขียนจะอธิบายให้ทราบว่าโค้ดบล็อกคืออะไร และนำไปใช้งานอย่างไรได้บ้าง จะเปรียบเทียบกับมาโคร (มาโครกล่าวไปแล้วในตอนที่ผ่านมา) ซึ่งสามารถจะใช้โค้ดบล็อกแทนมาโครได้และยังทำให้โปรแกรมสั้นลงและทำงานได้เร็วขึ้นด้วย

แบบของข้อมูลในคลิปเปอร์

ในคลิปเปอร์ 5.0 สนับสนุนแบบของข้อมูลดังนี้

- Array
- Character
- Code Block
- Numeric
- Date
- Memo
- NIL

ความจริงแล้วยังมีแบบข้อมูลชนิดวัตถุ (object) อีกชนิดที่คลิปเปอร์ 5.0 ก็สนับสนุนแต่ในคู่มือของคลิปเปอร์จะแยกออกไปจากกลุ่ม (มุมคลิปเปอร์จะนำมาเสนอในตอนต่อ ๆ ไป) จากแบบของข้อมูลถ้าเปรียบเทียบกับคลิปเปอร์ซัมเมอร์ '87 จะพบว่ามีแบบข้อมูลที่เพิ่มเข้ามาคือ โค้ดบล็อก และ NIL

ข้อมูลแบบอะเรย์นั้นมีคุณสมบัติพิเศษที่เพิ่มเติมจากอะเรย์ในซัมเมอร์ '87 ค่อนข้างมากทีเดียว โดยเฉพาะในเรื่องของชนิดข้อมูลที่จัดเก็บในอะเรย์ซึ่งในซัมเมอร์ '87 และในภาษาคอมพิวเตอร์ทั่วไปนั้นข้อมูลในอะเรย์จะต้องเป็นข้อมูลชนิดเดียวกันเท่านั้น แต่ในคลิปเปอร์ 5.0 อะเรย์สามารถจัดข้อมูลต่างชนิดต่างแบบกันได้ ซึ่งคุณสมบัติอันนี้ทำให้อะเรย์ในคลิปเปอร์ 5.0 คล้ายคลึงกับข้อมูลแบบเรคอร์ด (Record) ของภาษาปาสคาล หรือโครงสร้าง (Structure) ของภาษาซี ทำให้การใช้งานอะเรย์มีประสิทธิภาพมากขึ้น (เรื่องราวของอะเรย์จะเป็นเรื่องต่อไปที่มุมคลิปเปอร์จะนำเสนอ)

ข้อมูลตัวอักษร (Character) ในคลิปเปอร์ 5.0 นั้นสามารถใช้งานตัวอักขระรหัสแอสกีศูนย์ (chr(0)) ได้ด้วย ในที่นี้คือ สามารถพิมพ์ออกมาทั้งทางจอภาพและเครื่องพิมพ์ซึ่งเป็นสิ่งที่สำคัญ และในคลิปเปอร์รุ่นก่อนไม่มี chr(0) นั้นใช้มากในการส่งรหัสไปควบคุมเครื่องพิมพ์หรือในการสื่อสารข้อมูลกับอุปกรณ์อื่น ข้อมูลตัวอักษรที่เชื่อมต่อกันก็คือ ข้อความ (หรือสายวลี (String) ในภาษาคอมพิวเตอร์ทั่วไป) ซึ่งในคู่มือของคลิปเปอร์ระบุว่าความยาวของข้อความนั้นมีได้ถึง 65,535 ตัวอักษร หรือ 64 กิโลไบต์ลบออกด้วยหนึ่งตัวอักษรที่ใช้เป็นตัวอักษรปิดท้าย (null terminator) ซึ่งจากข้อความที่ระบุในคู่มือทำให้ผู้เขียนเข้าใจว่าวิธีการเก็บข้อมูลของตัวอักษรจะคล้ายกันกับในภาษาซีที่ใช้ตัวอักษรรหัสศูนย์เป็นตัวปิดท้าย แต่ผู้เขียนได้ทดสอบดูแล้วปรากฏว่าวิธีการเก็บข้อมูลข้อความไม่ใช่ลักษณะที่ระบุในคู่มือ ผู้เขียนเข้าใจว่ามีการเก็บตัวนับความยาวของข้อมูลข้อความด้วยจำนวนเต็มบวก โดยไม่สนใจ chr(0) ว่าจะอยู่ตัวตำแหน่งใดของข้อความ ผู้เขียนได้ทำการทดสอบโดยเขียนโปรแกรมสั้น ๆ ดังนี้

```
teststr := "Test1" + chr(0) + "Test2"
? teststr
? len (teststr)
Test1 Test2
11
```

ข้อมูลแบบวันที่ (Date) เป็นแบบข้อมูลของคลิปเปอร์ที่มีมานานแล้ว และมีประโยชน์มากในการประมวลผลที่เกี่ยวกับวันที่ตามปฏิทิน เช่น การคำนวณหาวันที่ที่ห่างจากวันนี้ไปอีก 10 วัน คือวันที่เท่าไร วัน/เดือน/ปีอะไร เป็นต้น

การกำหนดรูปแบบการรับค่าและการแสดงผลของวันที่สามารถทำได้ด้วยคำสั่ง SET DATE TO ตามด้วยรูปแบบของวันที่ที่นิยมใช้ในประเทศต่าง ๆ ดังตารางที่ 1

ตัวอย่างเช่น ต้องการแบบวันที่ตามประเทศอังกฤษ ซึ่งนิยมเอาวันที่ขึ้นก่อนแล้วตามด้วยเดือนและปี ก็สามารถสั่งงานได้ดังนี้

```
SET DATE TO BRITISH
```

หรือเราสามารถกำหนดรูปแบบแสดงผลเองได้ด้วยคำสั่ง SET DATE FORMAT TO แล้วตามด้วยข้อความที่ระบุรูปแบบซึ่งในข้อความดังกล่าวตัวอักษร d, m และ y หมายถึง วันที่ เดือน และปีตามลำดับ เช่น ต้องการรูปแบบวันที่ลักษณะนี้ 1993:03:30 สามารถกำหนดได้ดังนี้ SET DATE FORMAT TO "yyyy:mm:dd" เป็นต้น และอีกคำสั่งที่เกี่ยวข้องก็คือ คำสั่ง SET CENTURY ON/OFF เป็นการบังคับให้แสดงตัวเลขของปีครบหลักพันหรือไม่

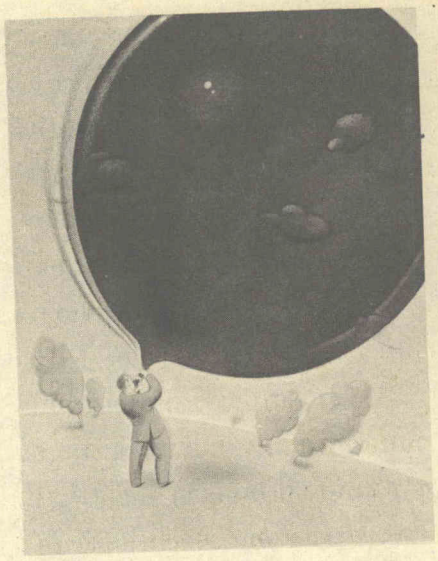
อีกสิ่งหนึ่งที่ควรทราบในการใช้งานตัวแปรวันที่ก็คือ คลิปเปอร์สามารถคำนวณวันที่ที่อยู่ในช่วงระหว่างวันที่ 01/01/0100 ถึงวันที่ 12/31/2999 เท่านั้น และค่าของวันที่ไม่สามารถเขียนตรงในลักษณะของ literal ได้ ซึ่งต้อง

ชื่อประเทศ	รูปแบบ
AMERICAN	mm/dd/yy
ANSI	yy.mm.dd
BRITISH	dd/mm/yy
FRENCH	dd/mm/yy
GERMAN	dd.mm.yy
ITALIAN	dd-mm-yy
JAPAN	yy/mm/dd
USA	mm-dd-yy

ตารางที่ 1 รูปแบบการแสดงผลวันที่ที่นิยมใช้ของประเทศต่าง ๆ

เครื่องหมาย	ความหมายการกระทำ
+	บวกวันที่เพิ่มด้วยค่าตัวเลข
-	ลบวันที่ออกด้วยค่าตัวเลขหรือวันที่
+ +	บวกเพิ่มอีก 1 วัน
- -	ลดลงอีก 1 วัน
= หรือ = =	เปรียบเทียบเท่ากัน
!=, <> หรือ #	เปรียบเทียบไม่เท่ากัน
<	เปรียบเทียบน้อยกว่า
<=	เปรียบเทียบน้อยกว่าหรือเท่ากัน
>	เปรียบเทียบมากกว่า
>=	เปรียบเทียบมากกว่าหรือเท่ากัน
= หรือคำสั่ง STORE	การกำหนดค่า
:=	การกำหนดค่าแบบอินไลน์
+=	การกำหนดค่าแบบอินไลน์พร้อมด้วยการบวกเพิ่ม
-=	การกำหนดค่าแบบอินไลน์พร้อมด้วยการหักออก

ตารางที่ 2
เครื่องหมายกระทำที่ใช้ได้กับข้อมูลวันที่



คำสั่งประกาศ	ผลของคำสั่งเมื่อไม่ได้กำหนดเริ่มต้นให้ตัวแปร
DECLARE	- กำหนดค่า NIL ให้กับสมาชิกของอะเรย์
LOCAL	- กำหนดค่า NIL ให้กับตัวแปรโลคอลและสมาชิกของอะเรย์
PARAMETERS	- กำหนดค่า NIL ให้กับพารามิเตอร์ที่ไม่ได้ถูกผ่านค่าให้ขณะที่มีการเรียกใช้ฟังก์ชันหรือโพรซีเดอร์
PRIVATE	- กำหนดค่า NIL ให้กับตัวแปรไพรเวตและสมาชิกของอะเรย์
PUBLIC	- กำหนดค่า NIL ให้กับสมาชิกของอะเรย์
STATIC	- กำหนดค่า NIL ให้กับตัวแปรสแตติกและสมาชิกของอะเรย์

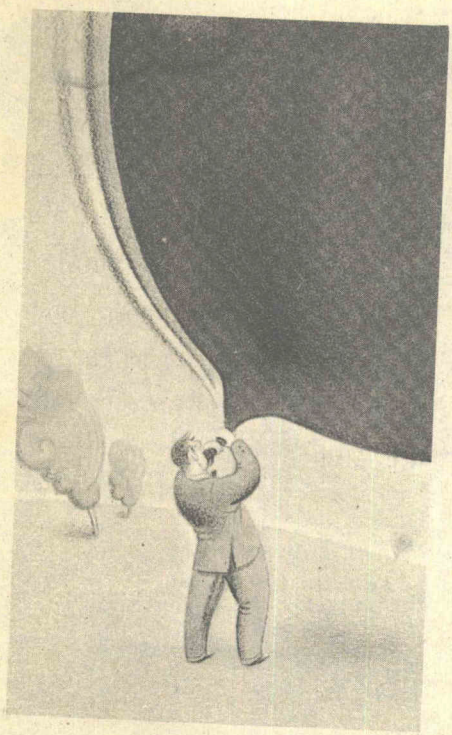
ตารางที่ 3
การประกาศตัวแปรกับการกำหนดตัว NIL ให้อัตโนมัติ

ใช้ฟังก์ชันทำการแปลงค่าจากข้อความเป็นวันที่คือ ฟังก์ชัน CTOD( ) ข้อมูลแบบวันที่สามารถนำมาคำนวณได้โดยใช้เครื่องหมายการกระทำ ดังตารางที่ 2

สำหรับกรณีข้อมูลแบบเมมโม (Memo) ใช้ได้กับฟิลด์ของแฟ้มข้อมูลเท่านั้น ซึ่งรายละเอียดของการใช้งานจะกล่าวถึงอีกครั้งหนึ่ง

แบบข้อมูลชนิด NIL เป็นข้อมูลแบบใหม่ที่เพิ่งมีเพิ่มเข้ามาในคลิปเปอร์ 5.0 เช่นกัน ข้อมูลแบบนี้มีเพียงค่าเดียวคือ NIL (ค่าของข้อมูลเป็นชื่อเดียวกับแบบข้อมูล) ซึ่งค่าของข้อมูลนี้มีไว้สำหรับจัดการกับตัวแปรที่ประกาศขึ้นมาแต่ยังไม่ต้องการกำหนดค่าขณะนั้น นอกจากนั้นค่านี้ยังใช้เป็นค่าที่กำหนดให้พารามิเตอร์ที่ไม่ได้ถูกส่งผ่านค่าเมื่อเวลาที่เรียกใช้ฟังก์ชันหรือโพรซีเดอร์ด้วย ตามปกติแล้วตัวแปรที่ถูกประกาศขึ้นมาด้วยคำประกาศ (เคยกล่าวไปแล้วในเรื่องของตัวแปร) โดยไม่ได้กำหนดค่าเริ่มต้น คลิปเปอร์จะกำหนดค่า NIL ให้กับตัวแปรเหล่านั้นอัตโนมัติ ดังตารางที่ 3

จากตารางที่ 3 พอสรุปได้ว่า การประกาศตัวแปรขึ้นด้วยคำสั่งทั้งหมดยกเว้นคำสั่ง PUBLIC และ FIELD จะมีการกำหนดค่า NIL ให้กับตัวแปรเหล่านั้นโดยอัตโนมัติหากการประกาศตัวแปรไม่มีการกำหนดค่าเริ่มต้นให้ การประกาศอะเรย์ก็เช่นกันไม่ว่าจะประกาศด้วยคำสั่งใดในตาราง รวมทั้งคำสั่ง PUBLIC จะมีการกำหนดค่า NIL ให้สมาชิกของอะเรย์ทุกสมาชิก ส่วนตัวแปรที่ถูกประกาศด้วยคำสั่ง PUBLIC แล้วไม่ได้กำหนดค่าเริ่มต้นให้ คลิปเปอร์จะกำหนดค่าเป็น .F. (เท็จ) ให้แทน ส่วนตัวแปรฟิลด์ที่ประกาศด้วย FIELD จะไม่มีการกำหนดค่าใด ๆ ให้ทั้งสิ้น (ให้ดูเรื่องการใช้งานตัวแปรในมุมคลิปเปอร์ตอนก่อน ๆ)



ในเรื่องของโพรซีเดอร์และฟังก์ชัน ผู้อ่านคงยังจำได้ว่าในการเรียกใช้งาน เราสามารถยกเว้นการส่งผ่านพารามิเตอร์ตัวใดก็ได้ ซึ่งหากตัวแปรพารามิเตอร์ใดที่ถูกยกเว้นการส่งผ่านจะมีค่าเป็น NIL

เอาล่ะครับ ผู้เขียนได้ทบทวนชนิดข้อมูลในบางส่วนที่เราอาจไม่เคยสนใจหรือใช้งาน ก็คงต้องมาเข้าเรื่องสำคัญกันได้แล้วครับ

โค้ดบล็อกคืออะไร?

ตอนนี้เราต้องมาทำใจกันให้ได้ก่อนว่า โค้ดบล็อก ก็คือข้อมูลชนิดหนึ่ง ดังนั้นสิ่งใดที่เราเคยทำกับข้อมูลชนิดอื่นได้ ก็สามารถจะทำกับโค้ดบล็อกได้เหมือนกัน เช่น กำหนดค่าให้กับตัวแปร ส่งผ่านค่าเป็นพารามิเตอร์ให้กับโพรซีเดอร์หรือฟังก์ชัน หรือใช้ส่งค่ากลับในกรณีเป็นฟังก์ชันที่ต้องมีค่าส่งกลับ และอื่น ๆ อีก และหากเราใช้ฟังก์ชัน TYPE ( ) หรือ VALTYPE ( ) กับตัวแปรที่เก็บโค้ดบล็อกเราจะได้ค่า "B" ออกมาจากฟังก์ชันทั้งสอง

โค้ดบล็อกจะไม่เหมือนกับข้อมูลชนิดอื่น ๆ ตรงที่เราไม่สามารถนำโค้ดบล็อกมาบวกหรือลบกันได้ แต่ในคลิปเปอร์ 5.0 จะมีฟังก์ชันพิเศษที่ใช้ทำงานกับโค้ดบล็อกโดยตรง ซึ่งเราจะดูต่อไป

เราไม่สามารถนำโค้ดบล็อกเก็บลงในแฟ้มข้อมูลได้ เพราะฟิลด์ในแฟ้มข้อมูลไม่มีฟิลด์ชนิดโค้ดบล็อก ดังนั้นโค้ดบล็อกจึงเป็นข้อมูลที่เก็บในตัวแปรในหน่วยความจำเท่านั้น แต่อย่างไรก็ตาม เราสามารถจะเก็บโค้ดบล็อกในลักษณะข้อความ ซึ่งสามารถเก็บลงแฟ้มข้อมูลและเมื่อต้องการใช้งานก็เปลี่ยนเป็นโค้ดบล็อกแล้วเรียกใช้งานได้ในภายหลัง

เมื่อทราบแล้วว่าโค้ดบล็อกเป็นข้อมูลชนิดหนึ่ง คราวนี้ก็มาถึงสิ่งที่ต้องทำความเข้าใจแล้วละครับว่า ข้อมูลแบบโค้ดบล็อกคืออะไร? ข้อมูลแบบโค้ดบล็อกความจริงแล้ว ก็คือ "ชุดคำสั่งหรือโปรแกรมสั้น ๆ ที่ถูกแปลแล้ว"

โค้ดบล็อกกับมาโคร

วิธีการที่ง่ายที่สุดที่จะอธิบายเรื่องโค้ดบล็อกให้เข้าใจ คือต้องเปรียบเทียบกับมาโคร ลองพิจารณาชุดคำสั่งต่อไปนี้

```
a = 1
b = 2
c = "a + b"
? & c
```

ถ้าหากยังจำเรื่องของมาโครได้ ก็จะทราบว่าคลิปเปอร์คำนวณและพิมพ์ค่า 3 ออกมา ซึ่งคือผลบวกของค่าในตัวแปร a และ b นั้นเอง ซึ่งเครื่องหมายกระทำมาโคร "&" จะนำข้อความ "a + b" ในตัวแปร c ไปวางต่อท้ายคำสั่ง ? ในกรณีนี้ข้อความ "a + b" จะถูกมองเป็น literal ดังนี้

```
? a + b
```

จากนั้นคลิปเปอร์จะทำการแปล (compile) คำสั่งนี้ก่อน แล้วจึงจะทำงานตามคำสั่งอีกที ซึ่งจะเห็นว่าการแปลคำสั่งต้องมีขั้นตอนการทำงานเท่ากับการแปลไฟล์โปรแกรมต้นฉบับ ดังนั้นจะเห็นว่าในโปรแกรมคลิปเปอร์ใด ๆ ที่ถูกเขียนขึ้นมาแล้วมีการใช้มาโครร่วมอยู่ด้วย การทำงานของโปรแกรมส่วนหนึ่งจะสูญเสียไปกับการแปลคำสั่งที่ได้จากการทำงานของมาโคร

จากการทำงานของมาโครดังกล่าว หากเราสามารถนำขั้น อนการแปลคำสั่งไปไว้ในช่วงของ รแปลไฟล์โปรแกรมต้นแบบ (source program) (ซึ่งโดยทั่วไปเราเรียกช่วงเวลานี้ว่า Compiled time ส่วนเวลาที่โปรแกรมทำงานเราเรียกว่า Run time) แทนที่จะอยู่ในเวลาที่โปรแกรมกำลังทำงาน ก็จะทำให้โปรแกรมมีการทำงานที่เร็วขึ้น แล้วจะทำอย่างไร? เราจึงสร้างชุดคำสั่งที่แปลแล้วเพื่อที่จะเรียกมาใช้งานในภายหลัง ลองพิจารณาชุดคำสั่งนั้น ๆ ดังนี้

```
a = 1
b = 2
c = {|| a + b}
? eval (c)
```

ค่าในตัวแปร c คือโค้ดบล็อก ซึ่งจะเห็นว่าเราสามารถสร้างโค้ดบล็อกขึ้นมาได้ด้วยรูปแบบพิเศษ โดยเริ่มต้นด้วยเครื่องหมายวงเล็บปีกกาเปิด { และปิดท้ายด้วยวงเล็บปีกกาปิด } ส่วนตัวอักษร | (อยู่บนแป้นพิมพ์ \) สองตัว

นั้นขอให้ข้ามไปก่อน ซึ่งจะบอกความหมายภายหลัง

สิ่งที่ควรทำความเข้าใจตอนนี้ก็คือ หลังจากเราแปล (หรือคอมพายล์) ชุดคำสั่งดังกล่าวข้างต้นและมีการทำงานของชุดคำสั่ง ค่าในตัวแปร c จะเก็บคำสั่ง (ซึ่งแปลแล้ว) ของการบวกค่า a และ b เข้าด้วยกัน และจากคำสั่งการกำหนดดังกล่าวชนิดของตัวแปร c จะเป็น "B" โค้ดบล็อก โปรดทำความเข้าใจในตอนนี้ว่า คำสั่ง c = {|| a+b} ไม่ใช่เป็นการเอาผลลัพธ์ของการบวก a และ b ไปเก็บใน c แต่จะเป็นการนำเอาคำสั่งที่แปลแล้ว (ซึ่งมีการทำงานโดยนำเอาค่าใน a และ b มาบวกกัน) เก็บเอาไว้ตัวแปร c

คราวนี้หากต้องการให้มีการทำงานตามคำสั่งในโค้ดบล็อกของตัวแปร c ก็สามารถทำได้ง่าย ๆ โดยการผ่านตัวแปร c เข้าไปในฟังก์ชัน EVAL ( ) ซึ่งเป็นฟังก์ชันใหม่ที่มีไว้ใช้งานกับโค้ดบล็อกโดยตรง ฟังก์ชัน EVAL ( ) จะเรียกโค้ดบล็อกทำงานและส่งค่าที่ทำงานได้กลับออกมา (Return) ซึ่งก็จะมีการทำงานคือ เอาค่าใน a และ b มารวมกัน และส่งผลลัพธ์ 3 ออกมา ถึงตอนนี้ก็อยากจะบอกท่านผู้อ่านว่าไม่ต้องกังวลใจกับฟังก์ชัน EVAL คิดเสียว่ามันก็คือฟังก์ชันที่เหมือนกับฟังก์ชันอื่นที่มีติดมากับคลิปเปอร์ เช่น ฟังก์ชัน STR ( ), ฟังก์ชัน DTOC ( ) ต่างออกไปเพียงรับพารามิเตอร์ คือ โค้ดบล็อกเท่านั้น

สรุปการสร้างโค้ดบล็อก

```
{|| นิพจน์ 1, นิพจน์ 2..... นิพจน์ n}
```

โค้ดบล็อกสามารถจะมีได้หลาย ๆ นิพจน์ (ดูรูปแบบการสร้างโค้ดบล็อก) โดยคั่นแต่ละนิพจน์ด้วย "," เช่น

```
cb = {|| a( ), b( ), c( )}
```

ถ้าหากเราส่งตัวแปร cb เป็นพารามิเตอร์ให้กับฟังก์ชัน EVAL การทำงานของคลิปเปอร์จะเรียกฟังก์ชัน a( ) ทำงานก่อนแล้วตามด้วยฟังก์ชัน b( ) และสุดท้ายคือ c( ) ซึ่งค่าที่ได้จากฟังก์ชัน c( ) จะถูกส่งกลับออกมา

เรามาสรุปกันอีกสักนิดถึงความแตกต่างระหว่างมาโครกับโค้ดบล็อก ก่อนจะผ่านไปยังเรื่องอื่น มาโครทำงานได้โดยการใช้เครื่องหมายมาโคร "&" ซึ่งการทำงานจะเกิดขึ้นตามลำดับต่อเนื่องกัน คือ ทำแปลคำสั่งก่อนแล้วทำงาน ซึ่งทั้งหมดเกิดขึ้นในขณะโปรแกรมทำงาน (Run time) แต่โค้ดบล็อกนั้นจะถูกแปลก่อนในขณะที่มีการแปลหรือคอมพายล์โปรแกรมต้นแบบ และจะถูกเรียกทำงานด้วยฟังก์ชัน EVAL( ) ซึ่งทั้งสองขั้นตอนจะแยกจากกัน และเราจะมาพิจารณาต่อไปโค้ดบล็อกช่วยเราอย่างไรบ้างในการเขียนโปรแกรม

โค้ดบล็อกช่วยลดเวลาการทำงานของโปรแกรม

การใช้มาโครขยายข้อความจะทำให้โปรแกรมทำงานช้าลงตามที่กล่าวมาแล้ว ลองพิจารณาชุดคำสั่งต่อไปนี้

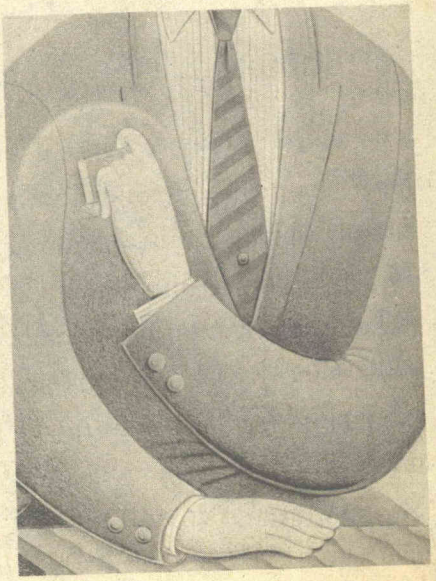
```
findit = "name #'Clipper'"
DO WHILE & findit
.
.
.
SKIP
ENDDO
```

สมมติชุดคำสั่งข้างต้นต้องการที่จะทำงานบางอย่างกับข้อมูลทุกเรคอร์ดที่ฟิลด์ name ไม่เท่ากับข้อความ "Clipper" ซึ่งถ้าหากว่ามีข้อมูลลักษณะดังกล่าวประมาณหนึ่งร้อยเรคอร์ด ทุก ๆ รอบของคำสั่ง DO WHILE ก็จะมีการทำงานตามเครื่องหมายมาโคร คือ แปลข้อความที่ระบุให้เปรียบเทียบฟิลด์ name กับข้อความ "Clipper" แล้วทำงานใหม่ ทำเช่นนี้ประมาณ 100 ครั้ง แต่ละครั้งมีการทำงานถึงสองขั้นตอนคือ แปลก่อนแล้วจึงทำงาน

คราวนี้หากเราเขียนชุดคำสั่งดังกล่าวใหม่ดังนี้

```
findit = {|| name = "Clipper"}
DO WHILE eval (findit)
.
.
.
SKIP
ENDDO
```

จะเห็นว่าคำสั่งการเปรียบเทียบค่าในฟิลด์ name กับข้อความ "Clipper" จะถูกแปลเพียงครั้งเดียวเมื่อแปลหรือคอมพายล์โปรแกรมต้นแบบ และทุกรอบของคำสั่ง DO WHILE จะมีเพียงการเรียกโค้ดบล็อกทำงานด้วยคำสั่ง



EVAL( ) เท่านั้น พอจะเห็นไหมครับว่า โค้ดบล็อกช่วยให้โปรแกรมทำงานเร็วขึ้นได้อย่างไรบ้าง

เรียกทำงานใหม่เมื่อไรก็ได้

เราสร้างเก็บโค้ดบล็อกเอาไว้ในตัวแปร แล้วเรียกขึ้นมาทำงานเมื่อไรก็ได้ ลองพิจารณาชุดคำสั่งต่อไปนี้

```
a = 1
b = 2
cb = {|| a + b}
? eval (cb)
a = 4
b = 3
? eval (cb)
```

เราจะเห็นว่า ตัวแปร cb เก็บชุดคำสั่งของการบวกค่าใน a และ b เข้าด้วยกัน ครั้งแรกที่เราเรียกโค้ดบล็อก cb ทำงาน ฟังก์ชัน EVAL( ) จะส่งค่า 3 กลับออกมาและเมื่อเราเปลี่ยนค่าใน a และ b โดยไม่ต้องเปลี่ยนแปลงในโค้ดบล็อก cb แต่เมื่อเรียกทำงานใหม่ ฟังก์ชัน EVAL( ) จะส่งค่า 7 ออกมาให้แทน

พารามิเตอร์ของโค้ดบล็อก

การใช้งานฟังก์ชันหรือโพรซีเดอร์นั้น ในขณะที่สร้างหรือประกาศฟังก์ชันหรือโพรซีเดอร์ เราสามารถระบุพารามิเตอร์ที่จะใช้เป็นตัวรับค่า (formal parameter) เมื่อมีการเรียกใช้งานจริงได้ ซึ่งเมื่อเรียกใช้งานจึงค่อยส่งค่าจริง (actual parameter) ให้ ซึ่งลักษณะการใช้งานดังกล่าวสามารถใช้กับโค้ดบล็อกได้ด้วยเช่นกัน

การระบุพารามิเตอร์ที่จะใช้เป็นตัวรับค่า (formal parameter) ในโค้ดบล็อกสามารถทำได้โดยใส่ชื่อพารามิเตอร์เหล่านั้นเอาไว้ภายในเครื่องหมาย || ดังตัวอย่าง

```
cb = {|a, b| a + b}
```

โค้ดบล็อกใน cb จะรับพารามิเตอร์สองตัว ซึ่งคล้าย ๆ กับการประกาศฟังก์ชันหรือโพรซีเดอร์ และพารามิเตอร์ที่เป็นตัวรับค่า และเช่นเดียวกันเมื่อเรียกใช้งานโค้ดบล็อกก็ต้องส่งผ่านค่าจริงไปให้โค้ดบล็อกเพื่อใช้ทำงานดังนี้

```
? eval (c, 7, 5)
```

ค่าพารามิเตอร์ทั้งสองค่า คือ 7 และ 5 จะถูกเก็บในพารามิเตอร์ a และ b ซึ่งก็ถูกนำมาบวกกันและส่งค่าคืนออกมาคือ 12 นั่นเอง

จากลักษณะของการส่งผ่านพารามิเตอร์ดังกล่าว หากในโค้ดบล็อกเป็นฟังก์ชัน เราก็สามารถจะส่งผ่านพารามิเตอร์ดังกล่าวให้ฟังก์ชันที่ถูกเรียกในโค้ดบล็อกได้ด้วย เช่น

```
cb = {|p1, p2| func(p1,p2)}
.
.
.
eval (cb, var1, var2)
```

ลักษณะนี้จะสะดวกกว่าการใช้มาโครมาก

สิ่งที่น่าสนใจอีกเรื่องหนึ่งก็คือ ตามปกติพารามิเตอร์ที่ผ่านเข้าไปในโค้ดบล็อกจะเป็นการส่งแบบส่งผ่านค่าอย่างเดียว (pass by value) ดังตัวอย่าง

```
a = 1
cb = {|pa| pa := 4}
eval (cb, a)
? a
```

จากการทำงานของชุดคำสั่งข้างต้น ใน a ยังคงมีค่าเป็น 1 เช่นเดิม แต่อย่างไรก็ตาม เราสามารถจะบังคับให้การส่งค่าเป็นแบบส่งผ่านที่อยู่ได้ (pass by adress) ดังนี้

```
a = 1
cb = {|pa| pa := 4}
evar (cb, @a)
? a
```

ซึ่งผลลัพธ์ในตัวแปร a จะมีค่าเป็น 4

สรุป

ในคลิปเปอร์ฉบับนี้ผู้เขียนได้แนะนำให้ผู้อ่านได้รู้จักว่าข้อมูลโค้ดบล็อกซึ่งเป็นข้อมูลแบบใหม่ในคลิปเปอร์ว่าคืออะไร? โดยผู้เขียนได้เปรียบเทียบกับมาโครให้ดู รวมทั้งผู้เขียนได้แนะนำการใช้งานในลักษณะง่าย ๆ ให้ด้วย ซึ่งผู้อ่านจะพบว่าโค้ดบล็อกนั้นมีประโยชน์ต่อการเขียนโปรแกรมคลิปเปอร์มากทีเดียว และหากสามารถใช้งานแทนมาโครได้ก็จะทำให้โปรแกรมที่ได้มีการทำงานที่รวดเร็วขึ้น ในตอนต่อไปผู้เขียนจะเสนอวิธีการใช้งานโค้ดบล็อกกับตัวแปรอะเรย์ และกับข้อมูลในแฟ้มข้อมูลว่าจะทำอย่างไร จึงไม่ต้องเขียนโปรแกรมลักษณะการวนรอบเพื่อกระทำข้อมูลกลุ่มอะเรย์ และกับข้อมูลในแฟ้มข้อมูลพบกันในตอนต่อไปครับ